SIGNAL89

Digital Signal Processing

For the TI-89

Author:

Lennart Isaksson

Table of contents

The reason this program exist

The main reason is to have a fast reachable "portable Matlab program" at hand any time.
The second reason, it's fun.

"I still don't have all the answer, but I'm beginning to ask the right questions."

What's New?

Version 2.4.0.		
Type	Name / prog / func	Description
New	Plotw()	A program to plot windows functions.
New	Boxcar()	Window functions for FIR filter design.
New	Triang()	Window functions for FIR filter design.
New	Bartlett()	Window functions for FIR filter design.
New	Blackman()	Window functions for FIR filter design.
New	Hamming()	Window functions for FIR filter design.
New	Hanning()	Window functions for FIR filter design.
New	Sinc()	Sinc.

Version 2.3.2.		
Type	Name / prog / func	Description
Changed	Documentation	Improved.
New	Getexpr()	Put the result in a vector.

Version 2.3.1.		
Type	Name / prog / func	Description
Changed	Help()	Improved.

Version 2.3.0.		
Type	Name / prog / func	Description
New	Filter()	Filter a signal.
New	Eye()	Identity matrix or flipped when minus.

Version 2.2.2.		
Type	Name / prog / func	Description
New	Xcorr2()	Correlation in 2-D.
New	FFTri()	New FFT and IFFT, code from "Numerical Recipes in C", Cambridge University Press. This brings out the real and imaginary part.
New	Help()	"Born to make you happy".

Changed	DFT2()	New name and from a program to a function.
Changed	IDFT2()	New name and from a program to a function.
Changed	DCT2()	New name and from a program to a function.
Changed	IDCT2()	New name and from a program to a function.
Changed	Xcorr()	From a program to a function.
Changed	MRound()	From a program to a function.
Changed	ZeroPad()	From a program to a function and some improvements.
Changed	Circ()	From a program to a function and some improvements.
Changed	Conv()	From a program to a function.
Changed	Conv2()	From a program to a function.
Changed	MENU89()	Some miner improvements.

Version 2.1.0.		
Type	Name / prog / func	Description
New	FFT()	Radix-2 Fast Fourier Transform, decimation in frequency. It's a function.
New	IFFT()	Radix-2 Inverse Fast Fourier Transform, decimation in frequency. It's a function.
Changed	DFT()	Discrete Fourier Transform 1-D. It's a function.
Changed	IDFT()	Inverse Discrete Fourier Transform 1-D. It's a function.
Changed	DFTG()	New name. Discrete Fourier Transform Graphic.
Changed	IDFTG()	New name. Discrete Fourier Transform (Graphic).
New	Flip()	Flips a matrix vertical and horizontal. It's a function.
New	FlipUD()	Flips a matrix horizontal. It's a function.
New	FlipLR()	Flips a matrix vertical. It's a function.
New	Rot90	Rotate a matrix 90 degree, counterclockwise. It's a function.
Changed	MENU89()	New name and some additions.
Changed	Mswap()	New name, Flip().

Version 2.0.0.		
Type	Name / prog / func	Description
Changed	Documentation	Documentation made in Microsoft Word.
New	DFT2D()	Discrete Fourier Transform in 2 dimensions.
New	IDFT2D()	Inverse Discrete Fourier Transform in 2 dimensions.
New	DCT2()	Discrete Cosine Transform in 2 dimensions.
New	IDCT2()	Inverse Discrete Cosine Transform in 2 dimensions.
New	xcorr()	Correlation of two signals, auto or cross, depend on the input signal.
New	Conv2()	Convolution with x and h in 2 dimensions.
New	MNU1()	Menu for the "Signal89".
New	mswap()	Flips the matrix vertical and horizontal.
New	mround()	Makes round inside the matrix.
New	zeropad()	Makes some zero padding round the matrix.

New	clellan()	Function, McClellan.
Changed	conv()	New name. Convolution, list is changed to matrix.
Changed	circ()	New name. Circular (convolution), list is changed to matrix.
Changed	dft()	List is changed to matrix and some of the graphics.
Changed	idft()	List is changed to matrix.

Installation of signal89

PC

- Start TI-Graph Link 89.
- Push on the icon  or menu "Link" and "Send…".
- Look for "Signal89.89g" and select it. Then push on the "OK" button.

TI89

- Run the program "Menu89" via "2nd" and "var-link".
- After that, show the new signal89 menu via "2nd" and "custom".
- Then push "F1" and "SetFold(signal)". If using older version than 2.4.0.

Definition of Fourier Transform Equation

Time domain

$$h(t) = \int_{-\infty}^{\infty} H(f) \cdot e^{2\pi i f t} df$$

$$h(n) = \frac{1}{N} \sum_{k=0}^{N-1} H(k) \cdot e^{2\pi i k n / N}$$

Frequency domain

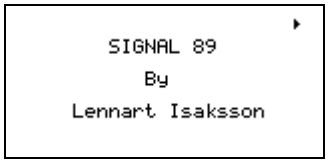
$$H(f) = \int_{-\infty}^{\infty} h(t) \cdot e^{-2\pi i f t} dt$$

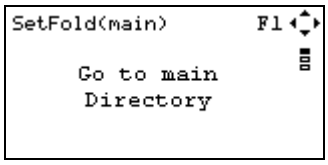
$$H(k) = \sum_{n=0}^{N-1} h(n) \cdot e^{-2\pi i k n / N}$$

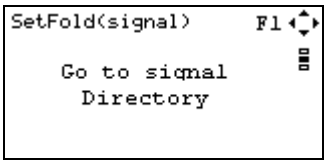
Programs and functions

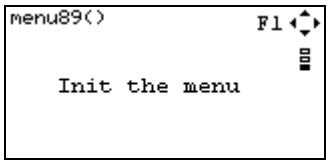
Help, Help( )

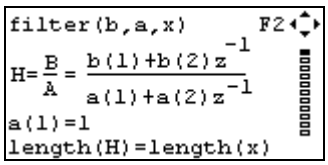
To make things easier.

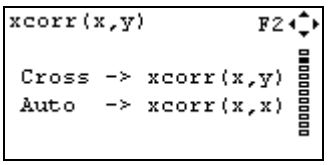


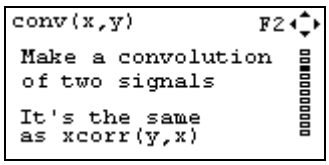


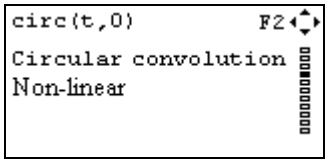


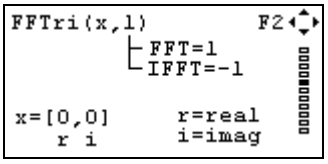


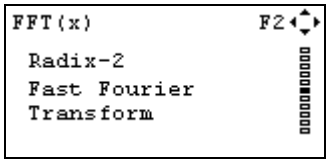


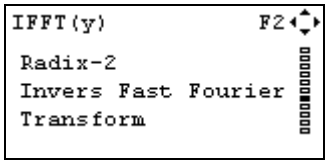


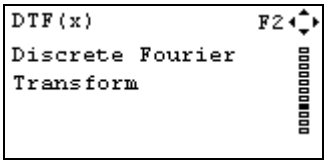


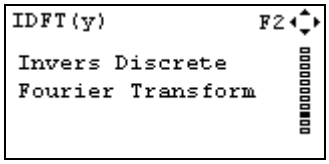


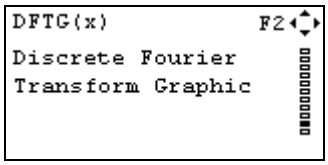


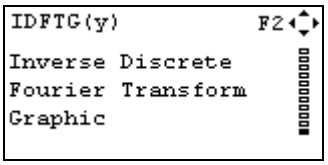


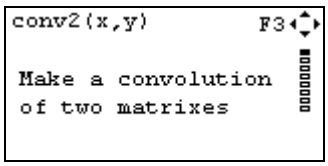


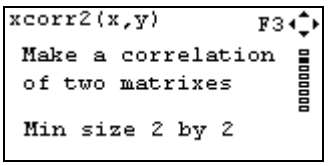


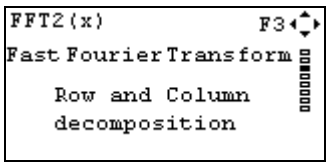


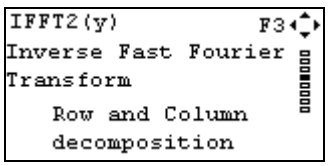


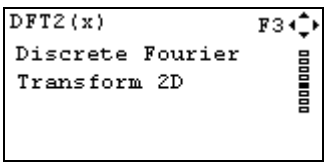


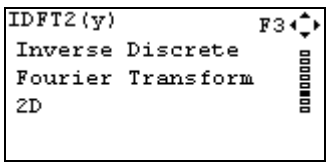












DCT2(x) F3
Discrete Cosine
Transform 2D

IDCT2(y) F3
Inverse Discrete
Cosine Transform 2D

Flip(F4
The Matrix
is flipped Vertical
and Horizontal

FlipUD(F4
The Matrix is
flipped horizontal

FlipLR(F4
The Matrix is
flipped vertical

Rot90(F4
Rotate 90 degree
counterclockwise

Eye(F4
Eye(2) [1 0]
[0 1]
Eye(-2) [0 1]
[1 0]

ZeroPad(x,3) F4
Zero padding a
Matrix to a
certain size

MRound(y,0) F4
Round off both real
and imag part

Boxcar(Num) F5

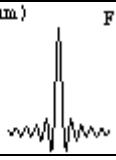
Triang(Num) F5


Bartlett(Num) F5


Blackman(Num) F5


Hamming(Num) F5


Hanning(Num) F5


sinc(Num,Num) F5


Help() F6
Born to make
you
happy

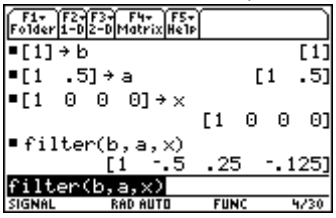
Filter, 1-D, filter(, ,)

This is a one dimensional digital filter. The filter is a “Direct Form II Transposed”.
If a[1] is not equal to 1, FILTER normalizes the filter coefficients by a[1].
The length of x is the same as the result.

$$H(z) = \frac{B(z)}{A(z)} = \frac{b(1) + b(2)z^{-1} + ... + b(nb+1)z^{-n}}{a(1) + a(2)z^{-1} + ... + a(na+1)z^{-n}}$$

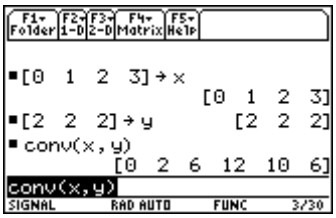
```
[1]->b
[1   .5]->a
[1   0   0   0]->x

filter(b,a,x)->[1   -.5   .25   -.125]
```

It’s a function, so the result is presented directly.



Convolution, 1-D, conv(,)

```
Make a convolution of two signals.
[0,1,2,3]->x
[2,2,2]->y
conv(x,y)

or

conv([0,1,2,3],[2,2,2])

It's a function, so the result is presented directly.
```



Convolution, 2-D, Conv2(,)

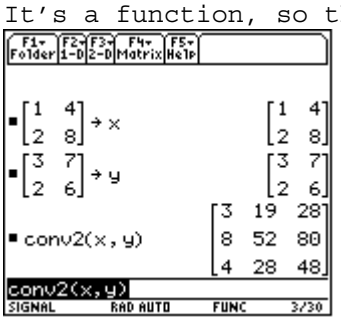
Make a convolution of two matrixes.

```
[1,4;2,8]->x
[3,7;2,6]->y
conv2(x,y)
```

or

```
conv2([1,4;2,8],[3,7;2,6])
```

It's a function, so the result is presented directly.



Considerations.
1. When using conv2(), both matrixes has to be the same size. If not, use the zeropad().
2. The minimum size of the matrix is 2 by 2.

Correlation , 1-D, Xcorr(,)

If you have two different signals then it's cross correlation. If you have the same signal then it's auto correlation.

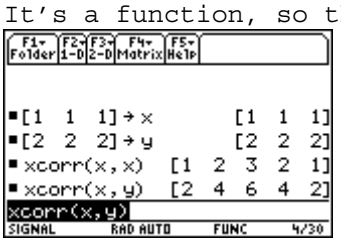
Auto correlation of a signal.

```
[1,1,1]->x
xcorr(x,x)

or

xcorr([1,1,1],[1,1,1])
```

Cross correlation of a signal.

```
[1,1,1]->x
[2,2,2]->y
xcorr(x,y)

or

xcorr([1,1,1],[2,2,2])
```

It's a function, so the result is there directly.



Correlation , 2-D, Xcorr2(,)

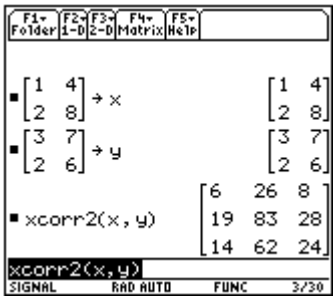
Make a correlation of two matrixes.

```
[1,4;2,8]->x
[3,7;2,6]->y
xcorr2(x,y)
```

or

```
xcorr2([1,4;2,8],[3,7;2,6])
```

It's a function, so the result is presented directly.



Considerations.
1. When using xcorr2(), both matrixes has to be the same size. If not, use the zeropad().
2. The minimum size of the matrix is 2 by 2.

Circular, 1-D, Circ(,)

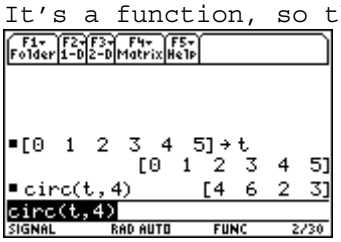
Circular (non linear), use the Convol() first.
The second parameter is the size of the vector. If it's set by zero it takes the length of the vector.

```
[0,1,3,5,3,0,0]->t
circ(t,4)
```

or

```
circ([0,1,3,5,3,0,0],4)
```

It's a function, so the result is presented directly.



Radix-2 Fast Fourier Transform, 1-D, FFT( )

This is a radix-2 Fast Fourier Transform decimation in frequency. This make a split X(k) into even- and odd number samples. The function is making an auto zero padding if necessary. See **Definition of Fourier Transform Equation.**
The matrix should be 2^x.

But what is a FFT anyway?
FFT is a DFT, but FFT is divided the calculation in a more intelligent way.

$$g_1(n) = x(n) + x\left(n + \frac{N}{2}\right) \qquad g_2(n) = \left(x(n) - x\left(n + \frac{N}{2}\right)\right) \cdot e^{\left(\frac{-i \cdot 2 \cdot \pi \cdot n}{N}\right)}$$

$$X(2 \cdot k) = \sum_{n=0}^{(N/2)-1} g_1(n) \cdot e^{\left(\frac{-i \cdot 2 \cdot \pi \cdot n \cdot k}{N/2}\right)} \qquad X(2 \cdot k + 1) = \sum_{n=0}^{(N/2)-1} g_2(n) \cdot e^{\left(\frac{-i \cdot 2 \cdot \pi \cdot n \cdot k}{N/2}\right)}$$

```
[0,1,2,3]->x
fft(x)

or

fft([0,1,2,3])
```

It's a function, so the result is presented directly.

```
[6  -2+2i  -2  -2-2i]
```

Radix-2 Inverse Fast Fourier Transform, 1-D, IFFT( )

This is a radix-2 Inverse Fast Fourier Transform decimation in frequency. This make a split X(k) into even- and odd number samples. The function is making an auto zero padding if necessary. See **Definition of Fourier Transform Equation.**

$$g_1(n) = x(n) + x\left(n + \frac{N}{2}\right) \qquad g_2(n) = \left(x(n) - x\left(n + \frac{N}{2}\right)\right) \cdot e^{\left(\frac{i \cdot 2 \cdot \pi \cdot n}{N}\right)}$$

$$X(2 \cdot k) = \frac{1}{N} \cdot \sum_{n=0}^{(N/2)-1} g_1(n) \cdot e^{\left(\frac{i \cdot 2 \cdot \pi \cdot n \cdot k}{N/2}\right)} \qquad X(2 \cdot k + 1) = \frac{1}{N} \cdot \sum_{n=0}^{(N/2)-1} g_2(n) \cdot e^{\left(\frac{i \cdot 2 \cdot \pi \cdot n \cdot k}{N/2}\right)}$$

```
[6,-2+2i,-2,-2-2i]->x
ifft(x)

or

ifft([6,-2+2i,-2,-2-2i])

It's a function, so the result is presented directly.
[0  1  2  3]
```

Radix-2 Fast Fourier Transform, 2-D, FFT2( )

Same as FFT( ) but with Row and Column decomposition.

Radix-2 Inverse Fast Fourier Transform, 2-D, IFFT2( )

Same as IFFT( ) but with Row and Column decomposition.

Radix-2 (with real and imaginary part) FFT and IFFT, 1-D, FFTri(,)

This is a radix-2 Fast Fourier Transform and Inverse Fast Fourier Transform decimation in frequency. (Originally it was a graphic program but I made it more general and a function). See **Definition of Fourier Transform Equation.**
As you can se, the code is excellent to use in C++, C or other program language.
(The original code is from "Numerical Recipes in C", Cambridge Univ Press, but it's wrong, so I made some changes to it. The error was then going from frequency domain back to the time domain, it didn't divided the amount of samples as you should do. Changed by Lennart Isaksson.)

```
r stands for real part.
i stands for imaginary part.
```

FFTri(x,1) FFT=1

```
[0,0,1,0,2,0,3,0]->x
 r i r i r i r i
FFTri(x,1)

or

FFTri([0,0,1,0,2,0,3,0],1)

It's a function, so the result is presented directly.
[6.00 0.00 -2.00 2.00 -2.00 0.00 -2.00 -2.00]
 r    i     r    i     r    i     r     i
```

FFTri(x,-1) IFFT=-1

```
[6.00 0.00 -2.00 2.00 -2.00 0.00 -2.00 -2.00]->x
 r    i     r    i     r    i     r     i
FFTri(x,-1)

or

FFTri([6.00 0.00 -2.00 2.00 -2.00 0.00 -2.00 -2.00],-1)

It's a function, so the result is presented directly.
[0.00 0.00 1.00 0.00 2.00 0.00 3.00 0.00]
 r    i    r    i    r    i    r    i
```

Discrete Fourier Transform, 1-D, DFT( )

Discrete Fourier Transform takes the signal from the time domain to the frequency domains. See **Definition of Fourier Transform Equation.**

```
[0,1,2,3]->x
dft(x)

or

dft([0,1,2,3])

It's a function, so the result is presented directly.
[6  -2+2i  -2  -2-2i]
```

Inverse Discrete Fourier Transform, 1-D, IDFT( )

Inverse Discrete Fourier Transform takes the signal from the frequency domains to the time domain. See **Definition of Fourier Transform Equation.**

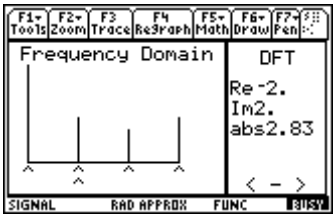
```
[6,-2+2i,-2,-2-2i]->x
dft(x)

or

dft([6,-2+2i,-2,-2-2i])

It's a function, so the result is presented directly.
[0  1  2  3]
```

Discrete Fourier Transform Graphic, 1-D, DFTG( )

Graphic interface.
Discrete Fourier Transform takes the signal from the time domain to the frequency domains. See **Definition of Fourier Transform Equation.**
But what is it?
Do you have a stereo with graphic equalizer? Yes, the bars that is jumping up and down is the values from DFT that's coming from your music cd. The result is the Frequency bars jumping up and down.

```
[0,1,2,3]->x
dft(x)

or

dft([0,1,2,3])
```

The result in variable y = [6 -2+2i -2 -2-2i]



Send the result in variable y to the variable t.

```
t->y

[1,1,0,0]->x
dft(x)

or

dft([1,1,0,0])
```

The result in variable y = [2 1-i 0 1+i]

Inverse Discrete Fourier Transform, 1-D, IDFTG( )

Inverse Discrete Fourier Transform takes the signal from the frequency domains to the time domains. See **Definition of Fourier Transform Equation.**

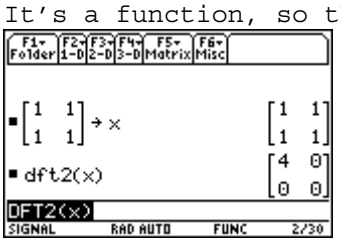
```
[12  4i  0  -4i]->y
idft(y)

or

idft([12  4i  0  -4i])

The result in variable t = [3  1  3  5]
```

Discrete Fourier Transform, 2-D, DFT2( )

Discrete Fourier Transform in 2 Dimensions. See **Definition of Fourier Transform Equation.**

```
[1,1;1,1]->x
dft2(x)

or

dft2([1,1;1,1])
```

It's a function, so the result is there directly.

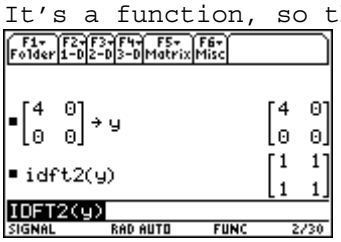


Inverse Discrete Fourier Transform, 2-D, IDFT2( )

Inverse Discrete Fourier Transform in 2 Dimensions. See **Definition of Fourier Transform Equation.**

```
[4,0;0,0]->y
idft(y)

or

idft([4,0;0,0])
```

It's a function, so the result is there directly.

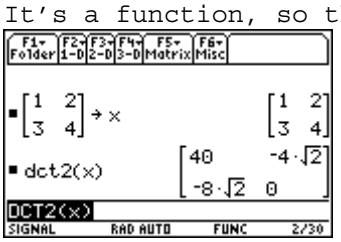


Discrete Cosine Transform, 2-D, DCT2( )

Discrete Cosines Transform in 2 Dimensions. See **Definition of Fourier Transform Equation.**

```
[1,2;3,4]->x
dct2(x)

or

dct2([1,2;3,4])
```

It's a function, so the result is there directly.



This is used frequently in compressing file format like "jpg".
The idea is to minimize the amount of information. In the above figure, variable x, we have four coefficients. Reduce the information by setting one element [1,2]=5.65685 to zero. Then you have a matrix like this:
[40,0;-11.3137,0]

If you then take the idct() to restore you picture, you are getting this:
[1.5,1.5;3.5,3.5]
As you have seen before on pictures, it got some square like pattern on the picture. Now you know why.

It's very near the optimum result you had before.
The optimum result is:
[1 2 3 4]

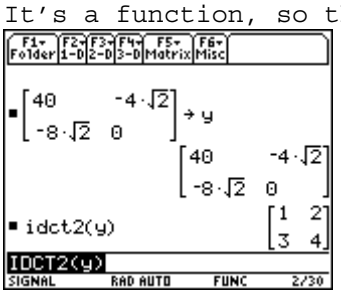
Matlab 5.x are using another definition.

Inverse Discrete Cosine Transform, 2-D, IDCT2( )

Inverse Discrete Cosines Transform in 2 Dimensions. See **Definition of Fourier Transform Equation.**

```
[40  -5.65685  -11.3137  0]->y
idct2(y)

or

idct2([40  -5.65685  -11.3137  0])
```

It's a function, so the result is there directly.



Get expression from the result, getexpr ( )

If you are using cSolve() to get the roots, the results is presented in a string form.

Solve:

```
cSolve(50x^4+35x^3+75x^2+20x+20=0,x)
```

you are getting:

```
s=-1/10+sqrt(39)/10*i or
s=-1/10-sqrt(39)/10*i or
s=-1/4+sqrt(15)/4*i or
s=-1/4-sqrt(15)/4*i
```

It's better to use Getexpr().

Solve:

```
Getexpr(cSolve(50x^4+35x^3+75x^2+20x+20=0,x))
```

You are getting the result in list form:

```
{-1/10+sqrt(39)/10*i,1/10-sqrt(39)/10*i,-1/4+sqrt(15)/4*i,1/4-sqrt(15)/4*i}
```

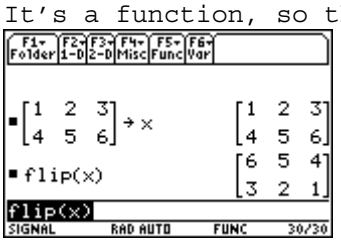
It's a better format.

Matrix Flip, 2-D, flip( )

The Matrix is flipped vertical and horizontal.

```
[1,2;3,4]->x
flip(x)

or

flip([1,2;3,4])
```

It's a function, so the result is there directly.

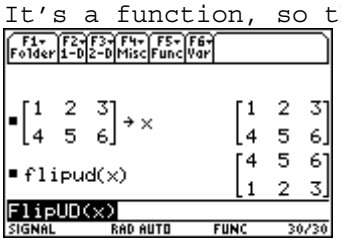


Matrix FlipUD, 2-D, flipud( )

The Matrix is flipped horizontal.

```
[1,2;3,4]->x
flipud(x)

or

flipud([1,2;3,4])
```

It's a function, so the result is presented directly.

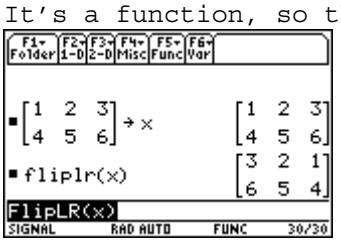


Matrix FlipLR, 2-D, fliplr( )

The Matrix is flipped vertical.

```
[1,2;3,4]->x
fliplr(x)

or

fliplr([1,2;3,4])
```

It's a function, so the result is presented directly.

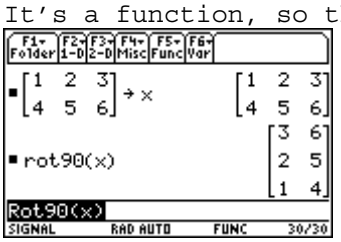


Matrix Rot90, 2-D, rot90( )

The Matrix is rotated 90 degree counterclockwise.

```
[1,2;3,4]->x
rot90(x)

or

rot90([1,2;3,4])
```

It's a function, so the result is presented directly.

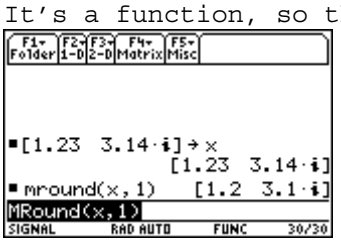


Matrix Round, 2-D, Mround(,)

The Matrix is round off both the real and imaginary part.

```
[1.23,3.14i]->x
mround(x,1)

or

mround([1.23,3.14i],1)
```

It's a function, so the result is presented directly.

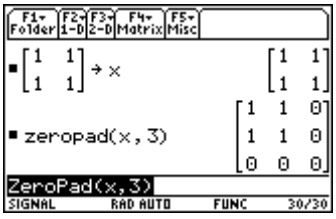


Matrix Zero Padding, 2-D, Zeropad(,)

Zero padding a matrix to a certain size.

```
[1,1;1,1]->x
zeropad(x,3)

or

zeropad([1,1;1,1],3)
```

It's a function, so the result is presented directly.



Eye, Identity Matrix , 2-D, Eye( )

Make an Identity matrix with a certain size. Yes, where is already a function named "indentity()", but this is better.

```
Eye(3)
```

It's a function, so the result is presented directly.

```
[1 0 0]
[0 1 0]
[0 0 1]
```

Special case,(this is not the identity matrix), but it's handy when flipping a matrix.

```
Eye(-3)
```

It's a function, so the result is presented directly.

```
[0 0 1]
[0 1 0]
[1 0 0]
```

Plotw(,)

A program to plot windows functions.
Plotw(50,1) Plots the function with 50 samples and a frame.

1=Frame
0=No frame

Boxcar( )

```
Boxcar(5)
```

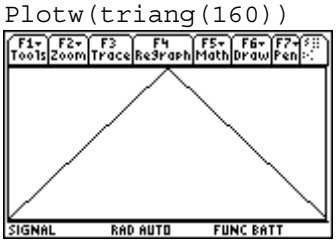
Answer:
```
[1 1 1 1 1]
```

Triang( )

```
Triang(5)
```

Answer:
```
[1/3 2/3 1 2/3 1/3]
```

```
Plotw(triang(160))
```

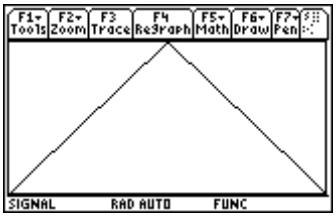


Bartlett( )

```
Bartlett(5)

Answer:
[0 1/2 1 1/2 0]
```

Observe that Bartlett have zero at element one and the last element compare to "Triang" function.

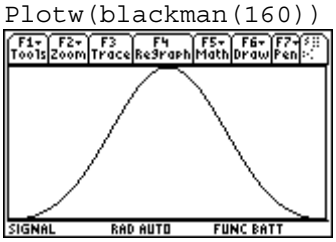
```
Plotw(bartlett(160))
```



Blackman( )

```
Blackman(5)

Answer:
[0 0.34 1 0.34 0]
```

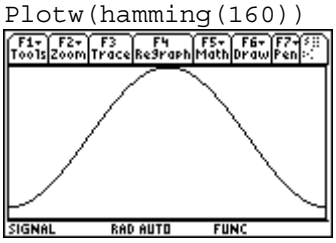
```
Plotw(blackman(160))
```



Hamming( )

```
Hamming(5)

Answer:
[0.08 0.54 1 0.54 0.08]
```

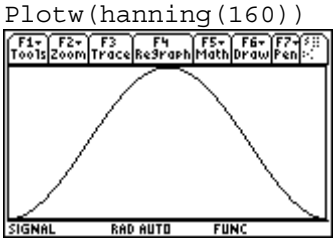
```
Plotw(hamming(160))
```



Hanning( )

```
Hanning(5)

Answer:
[0.25 0.75 1 0.75 0.25]
```

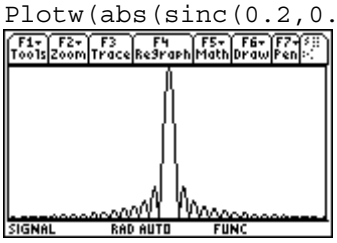
```
Plotw(hanning(160))
```



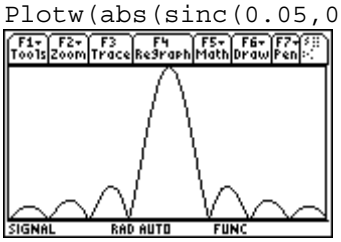
Sinc(,)

```
Sinc(0.4,1)

Answer:
[0 0.504551 0.935489 0.935489 0.504551 0]
```

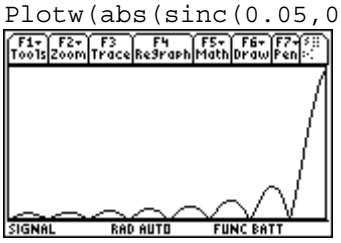
```
Plotw(abs(sinc(0.2,0.2*160*0.5)))
```




```
Plotw(abs(sinc(0.05,0.05*160*0.5)))
```




```
Plotw(abs(sinc(0.05,0.05*160)))
```

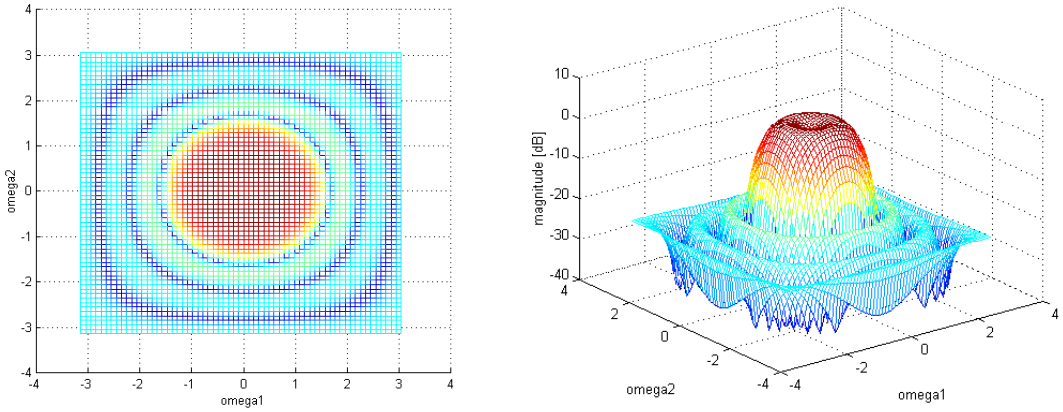


McClellan, 2-D, Clellan(,)

```
[0.2pi]->x
clellan(x,x)

or

clellan(0.2pi,0.2pi)

The result is 0.636271
```

McClellan is -0.5+0.5cos(w1)+0.5cos(w2)+0.5cos(w1)cos(w2).
As you can see, it's circular in the middle and squarer near the sides.

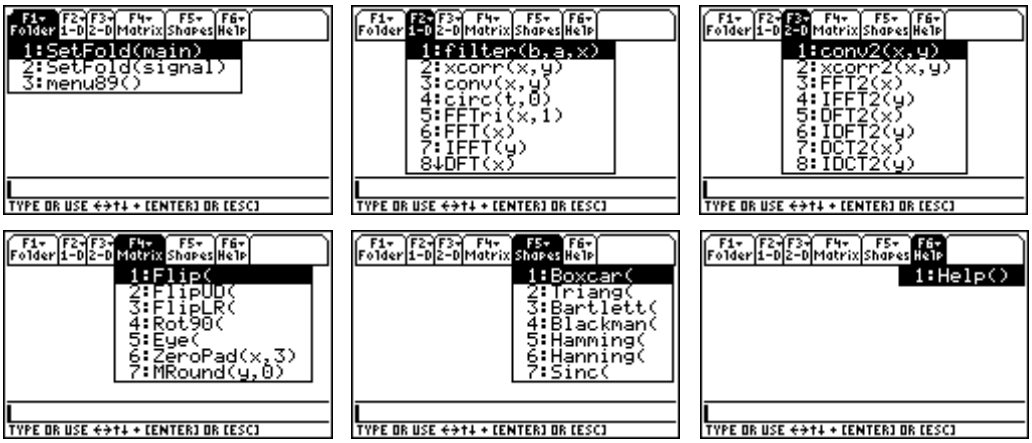
Figure below is a low pass filter with McClellan. 2-D FIR filter design using frequency transformation. 1-D FIR filter, using the transform T. Here is T the McClellan.



Figures above is made in Matlab.

Menu, menu89( )

```
Menu of Signal89.

Menu89()

Sets the custom menu.
```



Examples

How to use dft and idft instead of convolution.

The idea is to take x and h from the time domain to X an H in the frequency domain. Make the multiplication in the frequency domain. Go back to the time domain. Another way is to make a convolution in the time domain, for that you use the conv2().

We first starts to initialize two variables x and h.

```
[1,1;1,1]->x
[1,2;3,4]->h
```

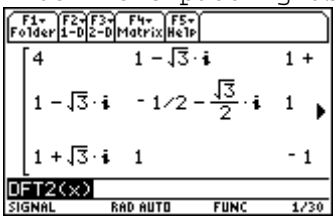
Take zeropad(x,3) (see zeropad) to make extra space in the matrix. It's necessary because otherwise the matrix is being circular convolution.

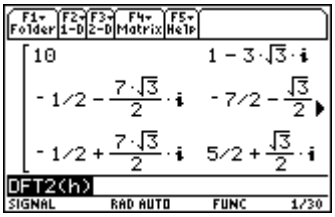
Considerations.

1. When multiply two matrixes, it has to be the same size and use the dot multiplication (.*).
2. Calculate the amount of zero padding by taking the size of matrix x adding the size of matrix h and reduce by one. In short, x+h-1. In our example, 2+2-1=3.

```
zeropad(x,3)->x
zeropad(h,3)->h
```

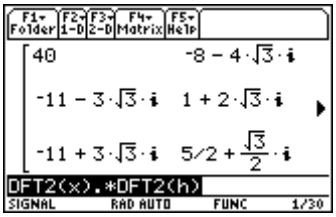
After zero padding use the dtf2d(x) and dtf2d(h).



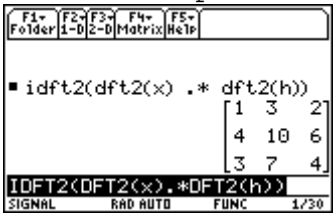


As you can see when going to the frequency domain, you have to expect imaginary number.

Take the two and make a dot multiplication.



The last step is to use the idft2d() to go back to the time domain.



How to make an imaginary partial fraction.

Say you have this equation to solve:

$$\frac{10s^2+3s+6}{50s^4+35s^3+75s^2+20s+20}$$

Use the cSolve() function in TI89 to solve the problem. The answer are in an unusable form, so use the function getexpr(), like this:

```
getexpr(cSolve(50s^4+35s^3+75s^2+20s+20=0,s))
```

Save the result in a variable, in this example, x. The roots of the characteristic equation are:

$$x[1]=-\frac{1}{10}+i\frac{\sqrt{39}}{10} \quad \text{rad/sec}$$

$$x^*[2]=-\frac{1}{10}-i\frac{\sqrt{39}}{10} \quad \text{rad/sec}$$

$$x[3]=-\frac{1}{4}+i\frac{\sqrt{15}}{4} \quad \text{rad/sec}$$

$$x^*[4]=-\frac{1}{4}-i\frac{\sqrt{15}}{4} \quad \text{rad/sec}$$

Now to the tricky part. Use the expand() function, like this:

```
expand((10/50*s^2+3/50*s+6/50)/((s-x[1])(s-x[2])(s-x[3])(s-x[4])))|s=s_
```

Yes, they're an underscore last, it's the trick. It solves it in an imaginary way fore you. The result should be:

$$\frac{\frac{4\sqrt{5}\sqrt{3}}{225}i}{s\_+\frac{1}{4}+\frac{\sqrt{5}\sqrt{3}}{4}i}+\frac{-\frac{4\sqrt{5}\sqrt{3}}{225}i}{s\_+\frac{1}{4}-\frac{\sqrt{5}\sqrt{3}}{4}i}+\frac{\frac{\sqrt{13}\sqrt{3}}{117}i}{s\_+\frac{1}{10}+\frac{\sqrt{13}\sqrt{3}}{10}i}+\frac{-\frac{\sqrt{13}\sqrt{3}}{117}i}{s\_+\frac{1}{10}-\frac{\sqrt{13}\sqrt{3}}{10}i}$$

So your residues are:

$$A_1 = i\frac{4\sqrt{15}}{225}$$

$$A_2 = -i\frac{4\sqrt{15}}{225}$$

$$A_3 = i\frac{\sqrt{39}}{117}$$

$$A_4 = -i\frac{\sqrt{39}}{117}$$

Author

Author: Lennart Isaksson
Country: Sweden
ICQ: 18869509
E-Mail: ets99lis@student.hk-r.se
Home page: http://www.student.hk-r.se/~ets99lis/

Any Questions or comments send an email or ICQ.

Credits

Mark Vulfson, USA, to a more compact code for dft() and idft().

Scott Campbell, USA, to converted FFTir() (Originally FFT) from TI83 to TI89.
The original code is from "Numerical Recipes in C", Cambridge Univ Press.

Reference

Digital Signal Processing, principles, algorithms, and applications
John G. Proakis
Dimitris G. Manolakis
ISBN 0-13-394289-9

Two-Dimensional, Signal and Image Processing
JAE S.LIM
ISBN 0-13-935322-4